

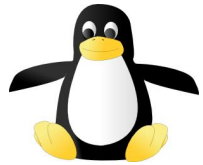
嵌入式Linux系统优化

Wu Zhangjin / Falcon
wuzhangjin@gmail.com

Tiny Lab – 泰晓实验室
http://tinylab.org

2013年6月4日

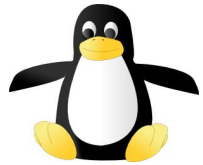




嵌入式Linux系统优化概要

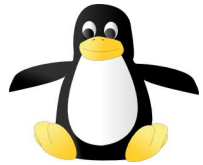
1. 增强系统稳定性
2. 加速系统和程序
3. 减小系统和程序大小
4. 降低系统功耗
5. 提高系统响应能力
6. 成本和组合优化效果
7. 终极优化策略





增强系统稳定性

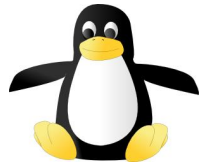




稳定性测试

- 测试目标
  - 做好最坏的预期
  - 模拟各种潜在使用场景
- 测试环境
  - 压力大
    处理器、内存、外设等各种资源利用率高
  - 覆盖范围广
    覆盖整个操作系统的方方面面
- 测试内容
  - 整个系统
    系统的兼容性（是否符合标准）、完整性（功能是否完善）、健壮性（是否容易Crash）
  - 单个功能
    具体某个外设和驱动是否正常工作



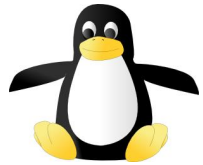


Linux标准测试套件

- Linux Test Project: http://ltp.sourceforge.net/
  - 可靠性、健壮性和稳定性
  - 覆盖测试：runalltest.sh
  - 压力测试：ltpstress.sh
- Open POSIX Test Suite: http://posixtest.sourceforge.net/
  - 一致性、功能、压力
  - make tests-pretty
- LSB Certification Tool: http://www.linuxfoundation.org/collaborate/workgroups/lsb
  - Linux App Checker
  - LSB Distribution Testkit
- ELC Platform Specification:

  http://www.opengroup.org/testing/testsuites/embedded.html
  - Embedded Linux Standard
- Phoronix Test Suite： http://www.phoronix-test-suite.com/
  - Hardware Platform Test Suite

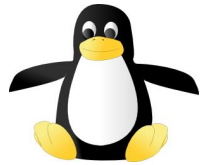




Linux硬件和驱动测试

- 处理器：cpuburn(FPU) + gkrellm + lm_sensors; stresslinux; burnCortex
- 图形处理器：Neocore (3D), NenaMark(OpenGPL ES 2.0)
- 内存：memtest86+, memtester
- 串口：minicom, cat /dev/ttyS0, cat /dev/ttyUSB0
- 闪存(NOR/NAND)：mtd-utils, flash_eraseall, flashcp
- 声卡：alsa-utils, amixer, alsamixer, aplay /dev/urandom; mplayer -ao alsa|oss
- 摄像头：v4l2, mplayer tv://dev/video0
- USB：usbutils, lsusb; usbstress, 可挂载存储设备测试
- 显卡/LCD: mplayer, directfb, directfb-examples

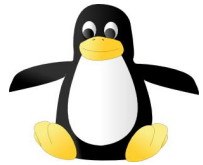




Linux硬件和驱动测试(cont.)

- I2C: i2c-tools, i2cdetect -l, i2cget, i2cset, i2cdump
- RTC时钟：util-linux, date + hwclock -r/-w
- 传感器：lm-sensors, sensors, gkrellm
- 键盘/鼠标：showkey; xdotool(simulate)
- 功能键：/sys/class/input/; HAL+Dbus, hotkey.py
- 触摸屏：Tslib
- 触摸板：gpointing-device-settings, tpconfig
- 磁盘：hdparm; hddtemp; badblocks -s -v /dev/sda1; smartctl -a /dev/ad0
- 以太网、无线、蓝牙：ethtool, dhclient, ifconfig, ping -I ethX -s psize -f, netperf -A -l, network-manager, wireshark

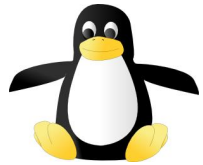




Linux硬件和驱动测试(cont.)

- 无线射频：rfkill; /sys/class/rfkill/rfkill0
- 看门狗：/dev/watchdog; kill watchdog daemon to see reset
- LCD背光：backlight; /sys/class/backlight
- 休眠、挂起：pm-utils; echo disk|standby|mem > /sys/power/state
- 处理器频率和电压调节：cpufreqd, powernowd; /sys/devices/system/cpu/cpu0/cpufreq
- 电池管理：gnome-power-manager, kpowersave; /sys/class/power_supply
- SPI/SD/MMC/PATA/SATA/PCI/PCIE：可挂载存储设备测试; dmesg,mount,umount,read,write,dd,tar

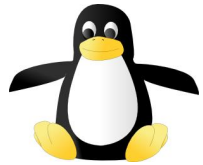




Defect报告和追踪

- bugzilla: http://www.bugzilla.org/
- Defect报告
  - 出错环境：内核配置、板子型号等
  - 出错症状：问题描述
  - 出错日志：Calltrace、Coredump等
  - 问题是否可重现、重现步骤
  - 可能解决办法：Workaround/Fixup
- Defect分析和调试
  - 见《Linux内核调试》
- Defect解决
  - 问题的详细分析过程
  - 问题的解决办法的详细描述
  - 进行回归性测试
  - 产生Patch并发送到邮件列表+详细的Review
  - 通过patchwork管理Patch
  - 把patch合并到源码仓库

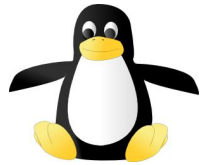




软件质量控制

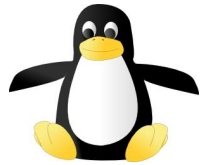
- 设计
  - 遵循标准规范, e.g: POSIX, LSB
- 流程
  - 遵循标准的软件开发流程
- 编码
  - 编码规范： Documentation/CodingStyle
  - 风格一致性： scripts/checkpatch.pl
- 静态检查
  - Sparse, coccinelle, smatch
  - more: http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
- 编译时检查
  - -Wall -Werror
  - -std=c89 -pedantic-errors
- 运行时诊断和调试
  - Debugging/Tracing API
  - Debugging/Tracing utilities





减少内核启动时间





CONFIG_PRINTK_TIME

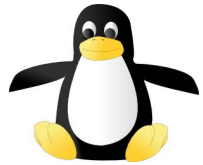
- 配置：Kernel hacking –> Show timing information on printks
- 结果：dmesg > boot.log

```
[    3.038027] Memory: 3154176k/3325940k available ...
[    3.042366] SLUB: Genslabs=13, HWalign=32, Order=0-3, MinObjects=0, CPUs=32, Nodes=1
[    3.050169] Hierarchical RCU implementation.
[    3.050558] RCU-based detection of stalled CPUs is enabled.
[    3.066428] console [ttyS0] enabled, bootconsole disabled
[    3.066965] Calibrating delay loop... 166.40 BogoMIPS (lpj=332800)
```

- 分析：scripts/show_delta boot.log | cut -d' ' -f2- | sort -k3 -gr

```
< 3.179683 >] 0000:01:00.0: eth0: (PCI Express:2.5GB/s:Width x1) ...
< 1.834118 >] Initializing cgroup subsys cpuset
< 1.555749 >] IP-Config: Complete:
< 0.463878 >] Memory: 3154176k/3325940k available ...
< 0.119374 >] Initrd not found or empty - disabling initrd
< 0.111845 >] Serial: 8250/16550 driver, 2 ports, IRQ sharing disabled
```





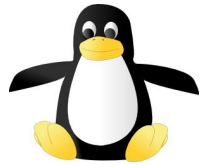
CONFIG_PRINTK_TIME(cont.)

- 避免丢失内核日志: CONFIG_LOG_BUF_SHIFT=21
- 在printk中打印时间戳： kernel/printk.c: vprintk()

```
if (printk_time) {
        ...
        t = cpu_clock(printk_cpu);
        nanosec_rem = do_div(t, 1000000000);
        tlen = sprintf(tbuf, "[%5lu.%06lu] ",
                        (unsigned long) t,
                        nanosec_rem / 1000);
        ...
}
```

- cpu_clock()调用sched_clock()
- 时间单位us，但有些平台精度只有10ms





CONFIG_PRINTK_TIME(cont.)

- 默认实现：直接通过jiffies转换
  - 如果HZ=100，jiffies单位为1/HZ, 即10ms
  - kernel/sched_clock.c: __weak sched_clock()

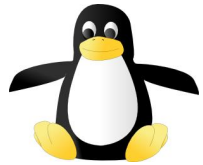
```
return (unsigned long long)(jiffies - INITIAL_JIFFIES)
                                    * (NSEC_PER_SEC / HZ);
```

- 高精度sched_clock()：读硬件时钟计数器
  - 如果时钟频率为400M，精度达2.5ns
  - arch/x86/kernel/tsc.c: native_sched_clock()

```
/* read the Time Stamp Counter: */
rdtscll(this_offset);

/* return the value in ns */
return __cycles_2_ns(this_offset);
```

- 潜在问题：计数器溢出？ include/linux/cnt32_to_63.h
- 相关信息：http://elinux.org/Printk_Times



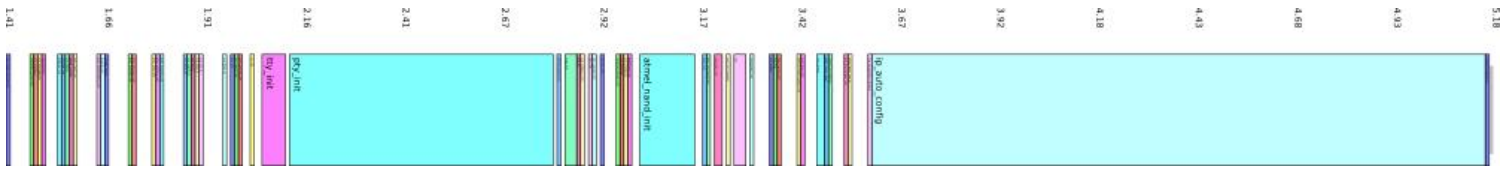


initcall_debug

- 可以跟踪initcalls, e.g. module_init()
- 启动：传内核参数initcall_debug和printk.time=1
- 结果：dmesg > boot.log

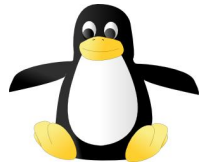
```
[   12.932000] calling  ip_auto_config+0x0/0x10e4 @ 1
[   14.516000] initcall ip_auto_config+0x0/0x10e4 returned 0 after 1540885 usecs
[   14.536000] async_waiting @ 1
[   14.536000] async_continuing @ 1 after 3 usec
[   26.308000] calling  inet6_init+0x0/0x4c4 [ipv6] @ 636
[   26.324000] initcall inet6_init+0x0/0x4c4 [ipv6] returned 0 after 16508 usecs
```

- 分析：scripts/bootgraph.pl boot.log > boot.svg



- SVG浏览器：firefox, chrome, inkscape



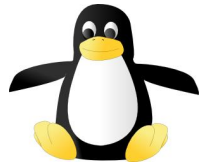


grabserial

- 下载: http://elinux.org/Grabserial
- 原理：给串口输出结果的每行加上时间戳
- 优点：
  - 不影响目标机器，完全在主机上运行
  - 可以记录任何来自串口的输出
- 用法： grabserial -v -d "/dev/ttyS0" -b 115200 -e 120 -t -m "Starting.*"

```
[    1.834118] Initializing cgroup subsys cpuset
[    1.838699] Linux version 2.6.34
[    2.049863] Initrd not found or empty - disabling initrd
[    6.646117] Switching to clocksource MIPS
[   13.354119] rtc-ds1374 1-0068: setting system clock ...
[   15.054502] Freeing unused kernel memory: 256k freed
[   15.338141] INIT: version 2.86 booting
[   16.773866] Starting udev: [  OK  ]
```





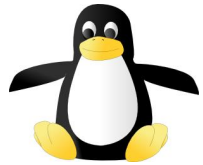
禁用内核的IP自动配置

- net/ipv4/ipconfig.c: ip_auto_config() 1.58s

```
/* Wait for devices to appear */
err = wait_for_devices();
if (err)
        return err;

/* Setup all network devices */
err = ic_open_devs();
if (err)
        return err;

/* Give drivers a chance to settle */
ssleep(CONF_POST_OPEN); /* 1s */
```

- 禁用办法
  - 不传递ip参数给内核
  - 禁用内核配置：CONFIG_IP_PNP*
- 延迟到用户态配置IP地址：/etc/init.d/rcS
- 思路：分开两个相互依赖的操作，消除不必要的IO等待时间

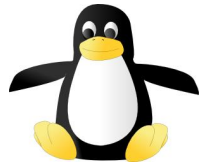




减少或禁用pseudo终端设备

- drivers/tty/pty.c: pty_init() 0.65 s
- pseudo终端：用于远程连接(ssh)或者X下的虚拟终端(xterm)
- 减少设备个数：LEGACY_PTY_COUNT=2
- 直接禁用：UNIX98_PTYS, LEGACY_PTYS
- 思路：根据实际需要减少或停用某些功能

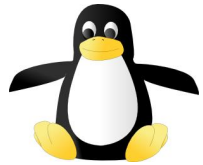




关闭控制台输出

- 打印日志信息到控制台很耗时
- 控制台设备：VGA、Framebuffer、串口
- 关闭控制台输出
  - 关闭部分输出: quiet参数, console_loglevel=4
  - 关闭所有输出：loglevel=0，副作用：错误也不显示
- 禁用控制台设备: e.g. 如果产品基于X11
  - CONFIG_{SERIAL*,VGA*, FB}=n
- 完全禁用printk：副作用：不能收集错误日志
  - CONFIG_{EARLY_PRINTK, PRINTK}=n
  - e.g. #define panic(...) do { } while (0) or loop or reboot
- 效果：提速30 ~ 60%
- 思路：考虑研发阶段和产品阶段的不同需求





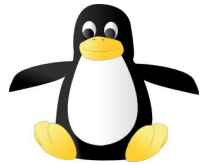
计算loops_per_jiffy

- {u,n}delay(): 根据loops_per_jiffy执行相应的nops
- loops_per_jiffy: 每个jiffy需要执行的nop次数
- 1个jiffy = 1/HZ = 10ms (HZ = 100)
  - tick_periodic() -> do_timer(1) -> jiffies_64 += 1;
- loops_per_jiffy计算：init/calibrate.c: calibrate_delay()
- 最慢：calibrate_delay_converge(): 250ms
- 最快：启动一次记录下来，下次直接传lpj参数给内核

```
$ dmesg | grep lpj
... calculated using timer frequency.. (lpj=7181976)
```

- 问题：下次启动时处理器主频变了怎么办?
- 解决办法：实现不基于loops_per_jiffy的{u,n}delay()
  - 新办法：读取硬件计数器直到delay的时长：e.g. delay_tsc()
  - 在delay_tsc()里头设置lpj_fine
- 思路：1、以静制动；2、转变思维方式寻求突破





采用更快的内核解压算法

- 为减少内核大小，一般都会对内核进行压缩

算法	内核大小	解压时间
不压缩	3.24M	-
LZO	1.76M	0.552s
Gzip	1.62M	0.775s
Bzip2	?	?
LZMA	1.22M	5.011s
XZ	?	?

表1: 不同压缩算法的解压速度比较

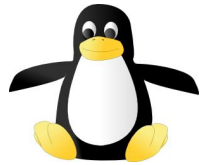
- 数据来源： http://free-electrons.com/blog/lzo-kernel-compression/
- 解压最快：LZO; 压缩比最高：XZ (针对可执行文件优化)
- 不压缩：拷贝时间？从哪里拷贝？多大？够小无须压缩。
- 思路：在不同需求之间进行权衡



减少内核大小

- 内核（未压缩的）越小，拷贝快，功能可能也少，执行快
- 减少内核大小的详细方法见《减少系统和程序的大小》
- 思路：启动速度的影响因素是多方面的

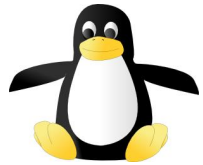




减少或者消除动态探测

- 基本思想：类似传lpj给内核避免计算loops_per_jiffy
- 思想扩展
  - 把“不变”参数作为binary，类似DTB，传给内核
  - 把“不变”参数定义成宏等数据，重编译内核，例如：arch/mips/include/asm/cpu-features.h
- 可采用的对象
  - 处理器：大部分属性都是固定的，比如tlbsize, cachesize等feature
  - PCI：PCI的外设不变的情况下，可把各外设的配置定死
- 效果：减少Probe过程，如果重编译，还可减少内核大小，减少大量分支跳转以及由此相应的分支预测失败等
- 思路：以静制动

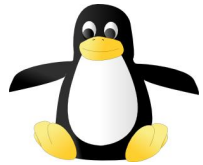




统筹考虑整个启动过程

- 一般启动过程：man boot
  - 硬件开机、重启、软件重启(reboot)
  - BIOS(EFI)、Boot Loader(U-boot)
  - OS Loader in MBR: Lilo, Grub
  - 装载Linux: cp.b, tftp
  - 启动Linux: boot, bootm
  - 运行Linux: 初始化、内核线程
  - 进入用户态
- 优化后
  - 硬件开机、重启、软件重启(reboot)
  - 完成必要的硬件初始化: X-loader
  - Kexecboot: 装载、启动、运行Linux并进入用户态
  - 切换其他Linux
- 效果：减少若干秒
- 思路：从整体上考虑问题





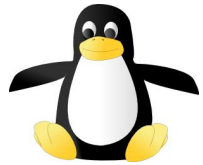
更多内核启动加速方法

- 快速重启：直接装载、启动、运行Linux并进入用户态
  - Kexec: Documentation/kdump/kdump.txt
  - reboot=soft (?)
- 快速分配内存： 内存预留(mem, reserve_bootmem)映射(ioremap)后直接使用
- 优化内存拷贝：DMA方式从Flash拷贝内核到RAM
- initcalls优化
  - 串行转并行：异步API: http://lwn.net/Articles/314808/
  - 延迟initcalls到用户态: http://elinux.org/Deferred_Initcalls
- 全速（或超速）启动、重启
  - 确保处理器在启动过程中全速运行，刚初始化时就设置处理器主频为全速
  - 在处理器启动过程中禁用变频、idle和节能模式
- 设备驱动特定的优化



减少系统启动时间
加快程序运行速度





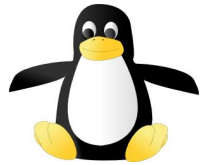
Init进程

- SysV init
  - 串行地启动预先配置好的服务
  - 启动下一个时需要等前一个完成
- upstart
  - 基于事件驱动，基于系统状态的改变启用和停用相应的任务
  - e.g. /etc/init/cron.conf

```
start on runlevel [2345]
stop on runlevel [!2345]
...
exec cron
```

- systemd
  - 基于socket和D-Bus激活来启动服务
  - 按需启动daemons，类似xinetd

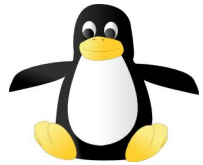




追踪Init启动服务过程

- 在内核态跟踪进程执行
  - 在系统调用kernel/exec.c: sys_exec()入口打印时间戳
  - 可以用scripts/show_delta统计分析
- bootchart
  - 统计资源利用率和各个进程的执行情况，导出SVG结果
  - 启动阶段资源利用率越高越好
- timechart
  - 统计更多信息，结果更详细，导出SVG结果
  - tools/perf/builtin-timechart.c

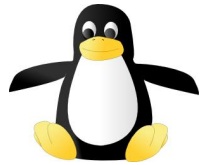




从休眠的映像文件启动内核

- 休眠接口：/sys/power/state
- 开发：启动内核和必须的应用，把系统休眠到磁盘或者Flash设备中，产生一个休眠映像文件
  - echo disk > /sys/power/state
- 产品：启动内核，直接从休眠映像文件恢复系统
- 效果：无需重新一个一个地启动程序，只需要恢复到一个早期休眠到内存的系统状态

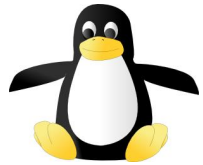




预读：readahead与tmpfs

- readahead
  - 预先读取文件到内存中
  - 减少iowait
  - 用法：sys_readahead(), readahead-list
- tmpfs
  - Documentation/filesystems/tmpfs.txt
  - tmpfs：内存中的文件系统+no swap
  - 如果内存足够可以考虑把程序预先复制到tmpfs中

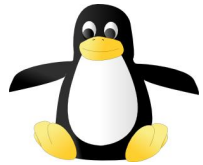




使用更快的文件系统

- 文件系统影响程序的IO操作
- Squashfs v.s. CramFS
- UBIFS v.s. JFFS2
- Reiser4 v.s. Ext3
- XFS(mount) v.s. JFS(cpu utilization)
- 文件系统操作属性：async,noatime,nodirtime,relatime
- 文件系统性能评测：Dbench, Bonnie++, IOzone, Flexible IO Tester





使用更小的执行文件

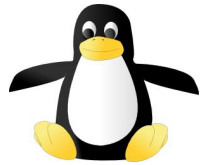
- 可执行文件更小，启动更快，占用内存更少
- ash v.s. bash
- busybox
- buildroot



编译器优化

- 常规：-O2, -O3
- gcc 4.5新特性：-flto
- 处理器特定优化：-march=, -mtune=
- 使用处理器优化指令: liboil
- 更多参数： http://en.wikipedia.org/wiki/Compiler_optimization

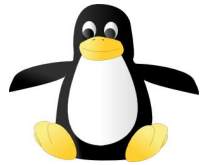




预链接

- 装载大量共享连接库很耗时
- 一般情况下可执行文件和共享连接库不变
- prelink通过修改可执行文件和共享连接库预先链接
- 减少动态链接的开销
- 更多信息：http://elinux.org/Pre_Linking





优化程序本身

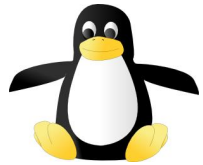
- 记录程序执行时间: time

```
$ time find /var/log/ -name "test" > /dev/null
real    0m0.006s
user    0m0.004s
sys     0m0.000s
```

- 跟踪程序函数执行开销: gprof
- 跟踪程序代码执行覆盖情况: gcov

```
        -:   35:int main(int argc, char **argv)
function main called 3 returned 100% blocks executed 60%
        3:   36:{
        3:   37:    if (argc < 2) {
    #####:   38:        a();
        -:   39:    } else {
        3:   40:        b();
        -:   41:    }
        -:   42:}
```

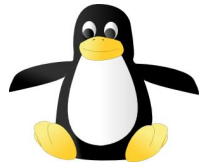




优化程序本身（cont.）

- 跟踪Cache miss, branch miss, page fault, tlb miss等：oprofile, perf, valgrind
  - Cache miss: cacheline对齐
  - branch miss: 消除不必要分支，通过gcov把执行多的branch调到前面或者用likely
  - page fault: mlock/munlock防止swap出去
  - tlb miss: 增加page大小



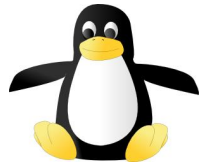


优化库函数: ltrace

- 用法： e.g. ltrace -T -f -o ltrace.log ls -l
- ltrace跟踪可执行文件调用的系统库文件
- pre_load(LD_PRELOAD)优化过后的函数，如memcpy

```
$ ltrace -c -f ls -l
% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 24.88    0.020998          42       491 strlen
 10.64    0.008981          41       216 __ctype_get_mb_cur_max
  9.02    0.007613          45       166 __overflow
  8.75    0.007388          47       156 __errno_location
  7.67    0.006472          41       156 memcpy
  5.33    0.004497          55        81 strcoll
       ...
------ ----------- ----------- --------- --------------------
100.00    0.084397                  1733 total
```





优化系统调用: strace

- 用法： e.g. strace -T -f -o strace.log ls -l
- 减少fork/exec，合并程序成applet
- 优化shell程序：去掉不必要的pipe, pipe也使用fork/exec
- 减少不必要的系统调用
- fast system call: e.g. MIPS syscall指令有预留的指令域，可以用于实现快速系统调用，比如模拟rdtsc，在用户态读取硬件时钟计数器

```
$ strace -f -ttt ls -l
1306002535.906459 execve("/bin/ls", ["ls", "-l"], [/* 56 vars */]) = 0
1306002535.907314 brk(0)                = 0x8c5a000
1306002535.907467 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
1306002535.907724 mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
1306002535.907950 access("/etc/ld.so.preload", R_OK) = 0
1306002535.908151 open("/etc/ld.so.preload", O_RDONLY) = 3
1306002535.908357 fstat64(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
1306002535.908554 close(3)              = 0
```



优化内核函数: Ftrace/Kgcov

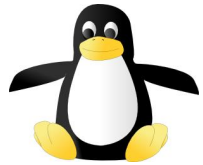
- Ftrace: Documentation/trace/ftrace.txt
  - Ftrace可以跟踪内核的函数执行情况
  - 执行程序前后开关Ftrace可追踪程序运行时的内核执行路径
  - 优化跟程序相关的内核路径

```
# tracer: function_graph
# CPU  DURATION                  FUNCTION CALLS
# |     |    |                    |   |   |   |
 0)               |  hrtimer_interrupt() {
 0)               |    __run_hrtimer() {
 0)               |      tick_sched_timer() {
 0)    5.723 us   |        do_timer();
 0)    1.518 us   |        hrtimer_run_queues();
 0)    1.552 us   |        run_posix_cpu_timers();
 0)    1.914 us   |        hrtimer_forward();
 0)  +28.661 us   |      }
 0)    2.290 us   |      enqueue_hrtimer();
 0)  +38.969 us   |    }
 0)  +67.588 us   |  }
```

- Kgcov: Documentation/gcov.txt
  - 内核代码覆盖率测试
  - lgcov：把测试结果转换成HTML格式方便浏览和分析

减少内核大小和内存使用

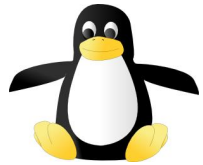




内核配置

- 默认关闭所有配置: make allnoconfig
- 开启一些必须的配置选项: lspci, lsusb...
- 通过CONFIG_EMBEDDED去掉某些功能: futex?
- 开启内核和initramfs压缩支持: lzo, lzma, gzip, bzip2, xz
- 采用支持压缩的文件系统: squashfs, ubifs
- 去掉内核调试支持: 调试功能、调试符号
- 去掉模块支持?
- -Os: CONFIG_CC_OPTIMIZE_FOR_SIZE
- strip -X: CONFIG_STRIP_ASM_SYMS

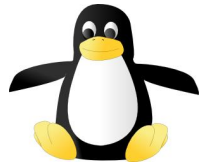




Linux-Tiny

- 目标：致力于降低内核大小和内存开销
- 下载：http://elinux.org/Linux_Tiny
- 策略
  - 让更多选项可配置
  - 删除内核消息(printk, BUG, panic, die)
  - 不内联inline函数：性能跟大小折中
  - 内存分配: Slob v.s. slab
  - 减少内存数据结果大小：性能与大小折中
  - 相同功能的简单实现：BFS v.s. CFS

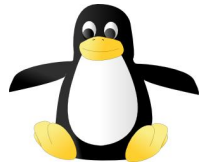




降低内存消耗

- initramfs v.s initrd
  - no block
  - no filesystem
  - no duplication

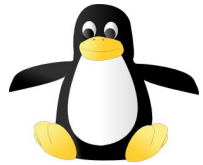




其他内核裁剪策略

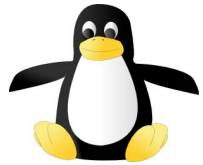
- strip -x: 删除non-global符号，模块的non-global符号可删除
- strip -s: 删除所有符号; strip -S: 删除跟调试相关符号
- sstrip(来自buildroot): 删除可执行文件的section table
- objcopy -O binary: 仅保留可直接执行的二进制映像
- section garbage collection patchset
  - -ffunction-sections -fdata-sections and -gc-sections
- 动态probing转静态definition
- 去掉更多的内核特性
  - 系统调用: ptrace
  - 内核和模块参数支持
  - 让某些宏可配置：NR_IRQS, COMMAND_LINE_SIZE...
  - 多选一：多个重复功能选其中一个，比如emulated FPU和hardware FPU
- 减少长调用：-mno-long-calls, 合并内核和模块空间





减少应用程序大小和内存使用



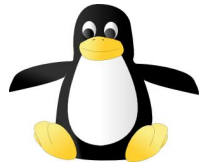


常见应用程序大小优化策略

- 静态链接v.s. 动态链接
- uclibc v.s eglibc v.s glibc
- strip, sstrip
- Library Optimizer: http://libraryopt.sourceforge.net/
- 编译器优化：-Os, -ffunction-sections -fdata-sections and -gc-sections
- 合并重复文件: dupmerge2,clink,finddup

降低系统功耗

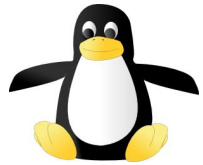




Tickless Kernel(Dynamic Ticks)

- 配置：CONFIG_NO_HZ
- HZ: 周期性的发出中断以便进行任务调度而支持多任务
- NO_HZ: 时钟中断按需发出, Idle时无时钟中断
- include/linux/clockchips.h: clock_event_device()
  - CLOCK_EVT_FEAT_ONESHOT
  - set_next_event
- 2.6.24: 支持X86, ARM, MIPS和PowerPC架构



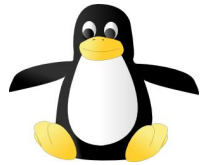


Powertop

- 监测频繁唤醒系统的内核和应用
- 提供一些减少功耗的建议

```
     PowerTOP version 1.13      (C) 2007 Intel Corporation

Cn                Avg residency       P-states (frequencies)
C0 (cpu running)        (28.9%)       Turbo Mode     18.7%
polling           0.0ms ( 0.0%)         1.80 Ghz      1.0%
C1 mwait          0.1ms ( 2.3%)         1200 Mhz      1.0%
C3 mwait          1.1ms (68.9%)          800 Mhz     79.3%


Wakeups-from-idle per second : 963.5    interval: 10.0s
no ACPI power usage estimate available

Top causes for wakeups:
  51.9% (455.2)   PS/2 keyboard/mouse/touchpad interrupt
  28.4% (249.4)   [kernel scheduler] Load balancing tick
   3.6% ( 31.5)D  firefox-bin
   4.6% ( 40.7)   [extra timer interrupt]
   2.5% ( 22.3)   soffice.bin

The program 'gimp' is writing to file '.gimp-2.6' on /dev/sda7.
This prevents the disk from going to powersave mode.

 Q - Quit   R - Refresh
```

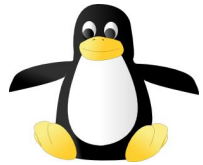




其他节能措施

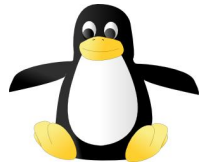
- CPUFreq
  - 处理器支持多级频率支持且可软件调节
  - 自动调节策略：governor用户态可配置
  - 实现驱动：配置可调节频率范围和操作底层寄存器
- 挂起隐藏的GUI
  - Suspend: kill -SIGTSTP <pid>
  - Resume: kill -SIGCONT <pid>
- 软件休眠与挂起
  - 休眠到Disk: echo disk > /sys/power/state
  - 挂起到内存：echo mem > /sys/power/state
  - 设备驱动支持：dev_pm_ops: suspend/resume
- 视频输出控制：video_output子系统
- 背光控制: backlight子系统
- 无线射频：rfkill子系统





提高系统响应能力





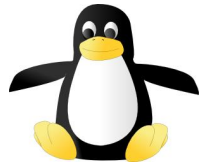
如何提高系统响应能力

- 测试系统响应延迟
  - cyclictest: git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rt-tests.git
- 更换调度策略
  - BFS v.s. CFS
  - 低延迟桌面：PREEMPT
- 中断线程化
  - 降低某些长中断处理的优先级
  - request_threaded_irq()
- 调整任务优先级：nice, chrt
- 绑定任务到处理器：taskset
- 资源分配：Session cgroup, ulimit



成本和组合优化效果



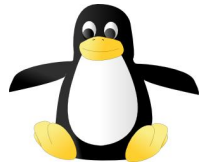


内存、存储、功耗与成本综合效果

	速度增加	内存减少	功耗降低	成本降低
更快速度			低功耗模式	便宜(慢)处理器
更少内存	快速分配 少swap 少flush cache		更少电耗	便宜内存 便宜处理器
更小空间	程序更快	内存少	片上功耗小	便宜存储设备
低功耗				便宜电池

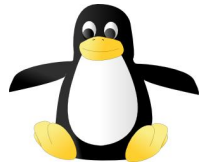
表2: 组合效果





终极优化策略





综合优化策略

- 详细的需求分析
- 好的设计与实现
- 功能与稳定性
- 系统优化